UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/696,828	10/30/2003	Joseph G. Laura	IDF 2506 (4000-14800)	1867

28003 7590 12/19/2008

SPRINT
6391 SPRINT PARKWAY
KSOPHT0101-Z2100
OVERLAND PARK, KS 66251-2100

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
12/19/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/696,828	LAURA, JOSEPH G.	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *22 September 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-3,6-8 and 10-29* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-3,6-8 and 10-29* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *20080922*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on September 22, 2008.

2. **Claims 1-3, 6-8, and 10-29** are pending.

3. **Claims 3, 8, and 24** has been amended.

4. **Claims 4, 5, and 9** have been canceled.

5. **Claims 27-29** have been added.

Response to Amendment

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. **Claims 1-3, 6-8, 10, 11, and 17-20** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 7,007,278 (hereinafter "Gungabeesoon")** in view of **US 6,757,746 (hereinafter "Boucher")**.

As per **Claim 1**, Gungabeesoon discloses:

- a memory block *(see Figure 1: 102)*;

- a COBOL program communicating with the memory block *(see Figure 1: 122; Column 11: 23-27, "... it is to be understood that the architecture but could also support legacy applications written in COBOL ...")*;

- a socket *(see Figure 6: 626A and 626B)*; and

- a COBOL routine writes information read from the socket to the memory block in response to a COBOL program call *(see Figure 6; Column 4: 53-58, "Operating system 120 and applications 122 reside in memory 102."; Column 11: 13-18, "The input data is then forwarded to socket or queue 626a as in step 642c, to the other application socket or queue 626b and I/O buffers if any and to the application runtime component 430, and eventually to the legacy program 122 that was waiting on a Read_Data method 640b.")*.

However, Gungabeesoon does not disclose:

- a COBOL routine callable from the COBOL program, the COBOL routine reads information from the socket, wherein the COBOL routine reads the information from the socket through a bit-level call to an operating system.

Boucher discloses:

- a socket routine callable from a program, the socket routine reads information from the socket, wherein the socket routine reads the information from the socket through a bit-level call to an operating system *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function." and "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105. When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete*

the connection setup. After setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived."). [Examiner's Note: Note that once the socket connection has been established, the socket function (the COBOL routine) maintains connection with the socket by interfacing with the kernel of the operating system. In other words, the socket function has to make "bit-level" calls to the operating system in order to interface with the operating system according to its kernel system call requirements. Thus, one of ordinary skill in the art would readily comprehend that in order for the socket function to interface with the operating system, it is inherent that the socket function must enable "bit-level" operating system calls, at least on the machine code level, to establish a socket connection.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include a COBOL routine callable from the COBOL program, the COBOL routine reads information from the socket, wherein the COBOL routine reads the information from the socket through a bit-level call to an operating system. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Gungabeesoon further discloses:

- wherein the COBOL program further communicates with the COBOL routine to initiate the COBOL routine communication with the socket and the memory block *(see Figure 6;*

Column 11: 8-10, "Subsequent interactions between the client interface on the network user agent 570 and the application 122 flows through the socket connections 626a and 626b.").

As per **Claim 3**, the rejection of **Claim 1** is incorporated; and Gungabeesoon further discloses:

- wherein the COBOL routine is further defined as one of a subroutine of the COBOL program, a COBOL library having a plurality of routines callable by the COBOL program, or a compiler enabled function usable by the COBOL program *(see Column 8: 14-17, "Each legacy application 122 has data 422 to be input/output to/from the application runtime operating system 430 according to the program I/O code 410 through the compiler runtime 420.").*

As per **Claim 6**, Gungabeesoon discloses:

- requesting, by a COBOL program stored on a computer readable medium, information from a socket *(see Column 11: 13-18, "The input data is then forwarded to socket or queue 626a as in step 642c, to the other application socket or queue 626b and I/O buffers if any and to the application runtime component 430, and eventually to the legacy program 122 that was waiting on a Read_Data method 640b.")*;

- writing, by the COBOL routine, information read from the socket to a memory block *(see Column 4: 53-58, "Memory 102 is a random-access semiconductor memory for storing data and programs ... Operating system 120 and applications 122 reside in memory 102.")*; and

- reading from the memory block, by the COBOL program, the information *(see Column 4: 53-58, "Memory 102 is a random-access semiconductor memory for storing data and programs ... Operating system 120 and applications 122 reside in memory 102.")*.

However, Gungabeesoon does not disclose:

- retrieving, by a COBOL routine stored on a computer readable medium, information from the socket through a bit-level call to an operating system.

Boucher discloses:

- retrieving, by a socket routine stored on a computer readable medium, information from the socket through a bit-level call to an operating system *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function." and "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105. When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete the connection setup. After setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived.")*. [Examiner's Note: Note that once the socket connection has been established, the socket function (the COBOL routine) maintains connection with the socket by interfacing with the kernel of the operating system. In other words, the socket function has to make "bit-level" calls to the operating system in order to interface with the operating system according to its kernel system call requirements. Thus, one of ordinary skill in the art would readily comprehend that in order for the socket function to interface with the

operating system, it is inherent that the socket function must enable "bit-level" operating system calls, at least on the machine code level, to establish a socket connection.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include retrieving, by a COBOL routine stored on a computer readable medium, information from the socket through a bit-level call to an operating system. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 7**, the rejection of **Claim 6** is incorporated; however, Gungabeesoon does not disclose:

- managing, by the COBOL routine, a connection with the socket.

Boucher discloses:

- managing, by the COBOL routine, a connection with the socket *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include managing, by the COBOL routine, a connection with the socket. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 8**, the rejection of **Claim 7** is incorporated; however, Gungabeesoon does not disclose:

- wherein managing includes one of listening on the socket connection or disconnecting the connection with the socket.

Boucher discloses:

- wherein managing includes listening on the socket connection *(see Column 16: 56-61, "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include wherein managing includes one of listening on the socket connection or disconnecting the connection with the socket. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 10**, the rejection of **Claim 6** is incorporated; however, Gungabeesoon does not disclose:

- establishing, by the COBOL routine, a connection with the socket.

Boucher discloses:

- establishing, by the COBOL routine, a connection with the socket *(see Column 3: 55-67 to Column 4: 1-4, "When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete the connection setup.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include establishing, by the COBOL routine, a connection with the socket. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 11**, the rejection of **Claim 10** is incorporated; however, Gungabeesoon does not disclose:

- wherein the connection with the socket is established in response to a request from the COBOL program.

Boucher discloses:

- wherein the connection with the socket is established in response to a request from the COBOL program *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include wherein the connection with the socket is established in response to a request from the COBOL program. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 17**, the rejection of **Claim 6** is incorporated; however, Gungabeesoon does not disclose:

- wherein the COBOL routine further includes a coordination module to coordinate such that the COBOL routine only reads when the socket has information and only writes when the socket is not full.

Boucher discloses:

- wherein the COBOL routine further includes a coordination module to coordinate such that the COBOL routine only reads when the socket has information and only writes when the socket is not full *(see Column 3: 55-67 to Column 4: 1-4, "After setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include wherein the COBOL routine further includes a coordination module to coordinate such that the COBOL routine only reads when the socket has information and only writes when the socket is not full. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 18**, the rejection of **Claim 6** is incorporated; however, Gungabeesoon does not disclose:

- initiating a call to the operating system by the COBOL routine to establish a socket connection.

Boucher discloses:

- initiating a call to the operating system by the COBOL routine to establish a socket connection *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include initiating a call to the operating system by the COBOL routine to establish a socket connection. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 19**, the rejection of **Claim 18** is incorporated; however, Gungabeesoon does not disclose:

- wherein the call to the operating system is further defined as a bit-level call to the operating system of a mainframe computer system.

Boucher discloses:

- wherein the call to the operating system is further defined as a bit-level call to the operating system of a mainframe computer system *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating*

system communication by calling the "socket" function." and "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105. When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete the connection setup. After setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived."). [Examiner's Note: Note that once the socket connection has been established, the socket function (the COBOL routine) maintains connection with the socket by interfacing with the kernel of the operating system. In other words, the socket function has to make "bit-level" calls to the operating system in order to interface with the operating system according to its kernel system call requirements. Thus, one of ordinary skill in the art would readily comprehend that in order for the socket function to interface with the operating system, it is inherent that the socket function must enable "bit-level" operating system calls, at least on the machine code level, to establish a socket connection.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include wherein the call to the operating system is further defined as a bit-level call to the operating system of a mainframe computer system. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

As per **Claim 20**, the rejection of **Claim 19** is incorporated; and Gungabeesoon further discloses:

- wherein the COBOL routine is further defined as written in COBOL programming language *(see Figure 1: 122; Column 11: 23-27, "... it is to be understood that the architecture but could also support legacy applications written in COBOL ...")*.

8. **Claims 12-14** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Gungabeesoon** in view of **Boucher** as applied to Claim 6 above, and further in view of **US 6,931,623 (hereinafter "Vermeire")**.

As per **Claim 12**, the rejection of **Claim 6** is incorporated; however, Gungabeesoon and Boucher do not disclose:

- wherein the COBOL routine provides an address to the COBOL program, the address identifying a location of the memory block where the information is written.

Vermeire discloses:

- wherein the COBOL routine provides an address to the COBOL program, the address identifying a location of the memory block where the information is written *(see Column 4: 35-44, "... a reference to the binary data contained within the record layout at the time the programming call to read or write data. The reference to the binary data is most likely a memory address (a "pointer") as implemented in most programming languages.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Vermeire into the teaching of Gungabeesoon to include wherein the COBOL routine provides an address to the COBOL program, the address identifying a location of the memory block where the information is written. The modification

would be obvious because one of ordinary skill in the art would be motivated to locate data in memory.

As per **Claim 13**, the rejection of **Claim 12** is incorporated; however, Gungabeesoon and Boucher do not disclose:

- mapping, by the COBOL program, the memory block into the COBOL program.

Vermeire discloses:

- mapping, by the COBOL program, the memory block into the COBOL program *(see Column 6: 43-55, "An existing COBOL copybook, an example of which is shown in FIG. 3, or a PL/I record definition in the source code of an existing legacy application are examples of a source record definition." and "The source record definition is processed by a lexical analyzer FIG. 2 capable of translating the language-specific representation of a record layout into a language-neutral and computer-architecture neutral representation of the data layout ("metadata"). This metadata is stored on a persistent storage medium 35 of FIG. 12 and accessed and managed via the workbench.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Vermeire into the teaching of Gungabeesoon to include mapping, by the COBOL program, the memory block into the COBOL program. The modification would be obvious because one of ordinary skill in the art would be motivated to locate data in memory.

As per **Claim 14**, the rejection of **Claim 13** is incorporated; however, Gungabeesoon and Boucher do not disclose:

- wherein the mapping is accomplished using a copybook.

Vermeire discloses:

- wherein the mapping is accomplished using a copybook *(see Column 6: 43-55, "An existing COBOL copybook, an example of which is shown in FIG. 3, or a PL/I record definition in the source code of an existing legacy application are examples of a source record definition.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Vermeire into the teaching of Gungabeesoon to include wherein the mapping is accomplished using a copybook. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the physical layout of data.

9. **Claims 15 and 16** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Gungabeesoon** in view of **Boucher** as applied to Claim 6 above, and further in view of **US 5,745,748 (hereinafter "Ahmad")**.

As per **Claim 15**, the rejection of **Claim 6** is incorporated; however, Gungabeesoon and Boucher do not disclose:

- wherein the information is provided in an EBCDIC format and wherein the method further comprises converting the information from the EBCDIC format to an ASCII format.

Ahmad discloses:

- wherein the information is provided in an EBCDIC format and wherein the method further comprises converting the information from the EBCDIC format to an ASCII format *(see Column 3: 18-21, "... if the data to be downloaded are in the EBCDIC format, as is common for mainframe computers, it must often be converted to the ASCII format for PC storage or use.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Ahmad into the teaching of Gungabeesoon to include wherein the information is provided in an EBCDIC format and wherein the method further comprises converting the information from the EBCDIC format to an ASCII format. The modification would be obvious because one of ordinary skill in the art would be motivated to store or use the information in a PC *(see Ahmad – Column 3: 18-21).*

As per **Claim 16**, the rejection of **Claim 15** is incorporated; however, Gungabeesoon and Boucher do not disclose:

- wherein the conversion is accomplished by the COBOL routine.

Ahmad discloses:

- wherein the conversion is accomplished by the COBOL routine *(see Column 3: 52-56, "... a system and method were needed to enable a mainframe-class application program under development in a PC-based COBOL development system to directly access data on a mainframe computer to which the PC was electronically linked.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Ahmad into the teaching of Gungabeesoon to

include wherein the conversion is accomplished by the COBOL routine. The modification would be obvious because one of ordinary skill in the art would be motivated to perform the conversion to allow access to mainframe computer data *(see Ahmad – Column 3: 18-21)*.

10. **Claims 21-23** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Gungabeesoon** in view of **Yu** and **Boucher**.

As per **Claim 21**, Gungabeesoon discloses:

- a memory block *(see Figure 1: 102)*;
- a COBOL program stored on a computer readable medium communicating with the memory block *(see Figure 1: 122; Column 11: 23-27, "... it is to be understood that the architecture but could also support legacy applications written in COBOL ...")*; and
- a COBOL routine writes information to the memory block in response to a COBOL program call *(see Figure 6; Column 4: 53-58, "Operating system 120 and applications 122 reside in memory 102."; Column 11: 13-18, "The input data is then forwarded to socket or queue 626a as in step 642c, to the other application socket or queue 626b and I/O buffers if any and to the application runtime component 430, and eventually to the legacy program 122 that was waiting on a Read_Data method 640b.")*.

However, Gungabeesoon does not disclose:

- a pipe; and

- a COBOL routine stored on a computer readable medium callable from the COBOL program, the COBOL routine reads information from the pipe, wherein the COBOL routine reads the information from the pipe through a bit-level call to an operating system.

Yu discloses:

- a pipe *(see Column 16: 4-9, "The kernel 70 opens up a PIPE and returns read and write file descriptors to communicate with the child process.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Yu into the teaching of Gungabeesoon to include a pipe. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a pipe called by a COBOL program.

Boucher discloses:

- a socket routine stored on a computer readable medium callable from a program, the socket routine reads information from a socket, wherein the socket routine reads the information from the socket through a bit-level call to an operating system *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function." and "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105. When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete the connection setup. After setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived.").* [Examiner's Note: Note that once the socket connection has been established, the socket function (the COBOL routine) maintains

connection with the socket by interfacing with the kernel of the operating system. In other words, the socket function has to make "bit-level" calls to the operating system in order to interface with the operating system according to its kernel system call requirements. Thus, one of ordinary skill in the art would readily comprehend that in order for the socket function to interface with the operating system, it is inherent that the socket function must enable "bit-level" operating system calls, at least on the machine code level, to establish a socket connection.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include a COBOL routine stored on a computer readable medium callable from the COBOL program, the COBOL routine reads information from the pipe, wherein the COBOL routine reads the information from the pipe through a bit-level call to an operating system. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a pipe called by a COBOL program.

As per **Claim 22**, the rejection of **Claim 21** is incorporated; and Gungabeesoon further discloses:

- wherein the memory block is further defined as a mainframe memory block and wherein the COBOL program and the COBOL routine are operable on a mainframe computer system *(see Figure 2: 202; Figure 6; Column 5: 45-46, "FIG. 2 is an example of a network server 200 which may access a legacy application stored on the computer 100.").*

As per **Claim 23**, the rejection of **Claim 22** is incorporated; however, Gungabeesoon and Boucher do not disclose:

- a create module communicating with a computer system to create a pipe connection;
- a connect module that promotes attachment to the pipe connection;
- an open module that opens the pipe connection to promote communication via the pipe connection;
- a write module that writes information to the pipe connection, the write module verifies that the pipe connection is not full prior to writing information and blocks when the pipe connection is full;
- a read module coupleable to the pipe connection to read information from the pipe connection;
- a release module to release the pipe connection;
- a remove module to remove the pipe connection from the computer system; and
- a delete module to delete the pipe connection wherein the pipe connection is closed.

Yu discloses:

- a create module communicating with a computer system to create a pipe connection *(see Figure 2: 200; Column 8: 12-15, "In using the socket interface, an application program invokes a socket function (block 200) which is typically processed as indicated in FIG. 2.")*;
- a connect module that promotes attachment to the pipe connection *(see Column 16: 56-61, "The other i/o socket functions not described (e.g. bind, listen, close, send, etc.) are processed in a manner similar to the above described socket functions. It will be appreciated that*

the non-blocking bind and listen socket functions typically are processed by server process 98 since they do not require a substantial amount of time to process.");

- an open module that opens the pipe connection to promote communication via the pipe connection *(see Column 16: 56-61, "The other i/o socket functions not described (e.g. bind, listen, close, send, etc.) are processed in a manner similar to the above described socket functions. It will be appreciated that the non-blocking bind and listen socket functions typically are processed by server process 98 since they do not require a substantial amount of time to process.")*;

- a write module that writes information to the pipe connection, the write module verifies that the pipe connection is not full prior to writing information and blocks when the pipe connection is full *(see Column 12: 26-29, "The application program uses the accepted socket to read and write data to and from the socket which connected to this socket and is not used to accept more connections.")*;

- a read module coupleable to the pipe connection to read information from the pipe connection *(see Column 12: 26-29, "The application program uses the accepted socket to read and write data to and from the socket which connected to this socket and is not used to accept more connections.")*;

- a release module to release the pipe connection *(see Column 16: 56-61, "The other i/o socket functions not described (e.g. bind, listen, close, send, etc.) are processed in a manner similar to the above described socket functions. It will be appreciated that the non-blocking bind and listen socket functions typically are processed by server process 98 since they do not require a substantial amount of time to process.")*;

- a remove module to remove the pipe connection from the computer system *(see Column 16: 56-61, "The other i/o socket functions not described (e.g. bind, listen, close, send, etc.) are processed in a manner similar to the above described socket functions. It will be appreciated that the non-blocking bind and listen socket functions typically are processed by server process 98 since they do not require a substantial amount of time to process.")*; and

- a delete module to delete the pipe connection wherein the pipe connection is closed *(see Column 16: 56-61, "The other i/o socket functions not described (e.g. bind, listen, close, send, etc.) are processed in a manner similar to the above described socket functions. It will be appreciated that the non-blocking bind and listen socket functions typically are processed by server process 98 since they do not require a substantial amount of time to process.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Yu into the teaching of Gungabeesoon to include a create module communicating with a computer system to create a pipe connection; a connect module that promotes attachment to the pipe connection; an open module that opens the pipe connection to promote communication via the pipe connection; a write module that writes information to the pipe connection, the write module verifies that the pipe connection is not full prior to writing information and blocks when the pipe connection is full; a read module coupleable to the pipe connection to read information from the pipe connection; a release module to release the pipe connection; a remove module to remove the pipe connection from the computer system; and a delete module to delete the pipe connection wherein the pipe connection is closed. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted using a pipe.

11. **Claims 24-26** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Gungabeesoon** in view of **Boucher** and **US 7,003,586 (hereinafter "Bailey")**.

As per **Claim 24**, Gungabeesoon discloses:

- writing, by a routine, information to an area *(see Column 4: 53-58, "Memory 102 is a random-access semiconductor memory for storing data and programs ... Operating system 120 and applications 122 reside in memory 102.")*; and

- reading, by a COBOL program stored on a computer readable medium, the information from the area, the COBOL program and the routine operating in the same runtime environment *(see Figure 1: 122; Figure 6; Column 4: 53-58, "Memory 102 is a random-access semiconductor memory for storing data and programs ... Operating system 120 and applications 122 reside in memory 102."; Column 11: 23-27, "... it is to be understood that the architecture but could also support legacy applications written in COBOL ...")*.

However, Gungabeesoon does not disclose:

- reading, by a routine stored on a computer readable medium, information from a socket through a bit-level call to an operating system, wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system.

Boucher discloses:

- reading, by a routine stored on a computer readable medium, information from a socket through a bit-level call to an operating system *(see Column 3: 55-67 to Column 4: 1-4,*

"In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function." and "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105. When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete the connection setup. After setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived."). [Examiner's Note: Note that once the socket connection has been established, the socket function (the COBOL routine) maintains connection with the socket by interfacing with the kernel of the operating system. In other words, the socket function has to make "bit-level" calls to the operating system in order to interface with the operating system according to its kernel system call requirements. Thus, one of ordinary skill in the art would readily comprehend that in order for the socket function to interface with the operating system, it is inherent that the socket function must enable "bit-level" operating system calls, at least on the machine code level, to establish a socket connection.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Boucher into the teaching of Gungabeesoon to include reading, by a routine stored on a computer readable medium, information from a socket through a bit-level call to an operating system. The modification would be obvious because one of ordinary skill in the art would be motivated to access data transmitted via a socket using a socket routine called by a COBOL program.

Bailey discloses:

- wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system *(see Column 2: 42-51, "System calls by the socket application 12 are intercepted by a sockets or virtual interface provider library (VIPL) applications programming interface (API) 16. Conventional kernel mode transition using existing OS resources involves passing the system call to a TCP/IP sockets provider 18 operating according to a user/kernel boundary 20."; Column 8: 12-16, "If the virtual address 44 is a value exceeding the prescribed offset, the memory controller 43 identifies the corresponding mapped physical 32-bit memory address 33 as a system memory mapped address 52 for accessing the corresponding queue pair buffer 24.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bailey into the teaching of Gungabeesoon to include wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system. The modification would be obvious because one of ordinary skill in the art would be motivated to ensure that the socket routine only access the prescribed operating system process *(see Bailey – Column 8: 17-24).*

As per **Claim 25**, the rejection of **Claim 24** is incorporated; and Gungabeesoon further discloses:

- wherein the area is a file *(see Column 11: 2-7, "The network page is populated with data from the data object as in step 652 ...").*

As per **Claim 26**, the rejection of **Claim 24** is incorporated; and Gungabeesoon further discloses:

- wherein the area is a memory area *(see Figure 1: 102)*.

12. **Claims 27 and 28** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Gungabeesoon** in view of **Boucher** as applied to Claims 1, 6, and 21 above, and further in view of **Bailey**.

As per **Claim 27**, the rejection of **Claim 1** is incorporated; however, Gungabeesoon and Boucher do not disclose:

- wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system.

Bailey discloses:

- wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system *(see Column 2: 42-51, "System calls by the socket application 12 are intercepted by a sockets or virtual interface provider library (VIPL) applications programming interface (API) 16. Conventional kernel mode transition using existing OS resources involves passing the system call to a TCP/IP sockets provider 18 operating according to a user/kernel boundary 20."; Column 8: 12-16, "If the virtual address 44 is a value exceeding the prescribed offset, the memory controller 43 identifies the corresponding mapped physical 32-bit memory address 33 as a system memory mapped address 52 for accessing the corresponding queue pair buffer 24.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bailey into the teaching of Gungabeesoon to include wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system. The modification would be obvious because one of ordinary skill in the art would be motivated to ensure that the socket routine only access the prescribed operating system process *(see Bailey – Column 8: 17-24)*.

Claim 28 is rejected for the same reason set forth in the rejection of Claim 27.

13. **Claim 29** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Gungabeesoon** in view of **Yu** and **Boucher** as applied to Claim 21 above, and further in view of **Bailey**.

As per **Claim 29**, the rejection of **Claim 21** is incorporated; however, Gungabeesoon, Yu, and Boucher do not disclose:

- wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system.

Bailey discloses:

- wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system *(see Column 2: 42-51, "System calls by the socket application 12 are intercepted by a sockets or virtual interface provider library (VIPL) applications programming interface (API) 16. Conventional kernel mode transition using existing OS resources involves passing the system call to a TCP/IP sockets*

provider 18 operating according to a user/kernel boundary 20."; Column 8: 12-16, "If the virtual address 44 is a value exceeding the prescribed offset, the memory controller 43 identifies the corresponding mapped physical 32-bit memory address 33 as a system memory mapped address 52 for accessing the corresponding queue pair buffer 24.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bailey into the teaching of Gungabeesoon to include wherein the bit-level call to the operating system includes correct bits, offsets, and memory mapping to sufficiently interface with the operating system. The modification would be obvious because one of ordinary skill in the art would be motivated to ensure that the socket routine only access the prescribed operating system process *(see Bailey – Column 8: 17-24).*

Response to Arguments

14. Applicant's arguments filed on September 22, 2008 have been fully considered, but they are not persuasive.

In the Remarks, Applicant argues:

a) As shown above, Boucher describes an application named Samba involved in the transmission of information between hosts. But the Samba application is a modern application that is natively network-aware. Samba is an open source implementation of networking protocols to share files between computers. Samba is natively able to call socket functions without any outside software components providing support. In contrast, claim 1 requires, "a COBOL routine callable from the COBOL program." The Samba application is not a COBOL program and the

socket function used by the Samba application is not a COBOL routine. In particular, as a natively network-aware application, Samba does contain socket functionality. But Samba's socket functionality is not a COBOL routine. Boucher only discloses in passing that an application may call a socket function which may cause an operating system to perform some function. Boucher contains no reference to the COBOL programming.

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, without acquiescing to the Applicant's assertion that the Samba application is not a COBOL program and the socket function used by the Samba application is not a COBOL routine, as previously pointed out in the Non-Final Rejection (mailed on 06/23/2008) and further clarified hereinafter, the Examiner first submits that Boucher is relied upon by the Examiner for its specific teaching of "a socket routine callable from a program, the socket routine reads information from the socket, wherein the socket routine reads the information from the socket through a bit-level call to an operating system." The § 103 rejections of the independent claims (Claims 1, 6, 21, and 24) made in the Non-Final Rejection (mailed on 06/23/2008) clearly states that Boucher discloses "a socket routine callable from a program, the socket routine reads information from the socket, wherein the socket routine reads the information from the socket through a bit-level call to an operating system." For example, by way of illustration, the particular section of the § 103 rejection of Claim 1 that is of interest is reproduced below with emphasis added for the Applicant's convenience:

However, Gungabeesoon does not disclose:

- a COBOL routine callable from the COBOL program, the COBOL routine reads information from the socket, wherein the COBOL routine reads the information from the socket through a bit-level call to an operating system.

Boucher discloses:

- **a socket routine callable from a program, the socket routine reads information from the socket, wherein the socket routine reads the information from the socket through a bit-level call to an operating system** *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function." and "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105. When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete the connection setup. After setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived." Note that once the socket connection has been established, the socket function (the COBOL routine) maintains connection with the socket by interfacing with the kernel of the operating system. In other words, the socket function has to make "bit-level" calls to the operating system in order to interface with the operating system according to its kernel system call requirements.)*.

Thus, the Examiner acknowledges that Boucher does not teach a COBOL routine and a COBOL program, but rather a socket routine and a program. Gungabeesoon teaches a COBOL program and sockets. However, Gungabeesoon does not teach a COBOL routine callable from the COBOL program. Examiner relies upon Boucher to teach a socket routine callable from a program; therefore, concluded that calling a socket routine from a program to retrieve data from a socket is a well-known concept within the computing art. Thus, one of ordinary skill in the art would readily recognize that the claimed feature of a "COBOL routine" is a socket routine written in the COBOL language. In addition, the plain language of the claims further requires that "the COBOL routine reads information from the socket." Again, one of ordinary skill in the art would readily recognize that the claimed feature of a "COBOL routine" is a socket routine written in the COBOL language because only a socket routine can read information from a

socket. Therefore, in making the § 103 rejection, the Examiner has established a *prima facie* case of obviousness by providing a rationale and a motivation to modify Gungabeesoon in view of the combined teachings of the prior art references to access data transmitted via a socket using a socket routine called by a COBOL program.

Second, Examiner further submits that Gungabeesoon is within the field of the Applicant's endeavor and hence is analogous prior art because Gungabeesoon's invention is directed to accessing legacy application programs, such as COBOL programs, over a computer network, such as the Internet. Boucher is concerned with the same problem which the Applicant sought to be solved and hence is analogous prior art because Boucher's invention is directed to protocol processing for information communicated between hosts such as computers connected to a network using sockets. Therefore, it is permissible to combine the teaching of Boucher into the teaching of Gungabeesoon to include the limitations disclosed by Boucher since Boucher provides a reason for combining the elements in the manner claimed. See MPEP § 2141.01(a).

Therefore, for at least the reasons set forth above, the rejections made under 35 U.S.C. § 103(a) with respect to Claims 1, 6, and 21 are proper and therefore, maintained.

In the Remarks, Applicant argues:

b) According to the ordinary meaning of the terms, the limitation of "bit-level" should be interpreted as on a position in scale or rank with bits (e.g. computer code representations of a binary digit such as a "1" or "0"). Therefore, according to the ordinary meanings of the terms, a "bit-level call" should be interpreted as a call made on a position in scale or rank with bits. In other words, a call including a sequence of l's and 0's. Applicant notes that this is consistent with

the specification in Paragraph 041. Applicant further notes that a high-level or assembly language call is not a call including a sequence of l's and 0's.

Accordingly, the socket function disclosed by Boucher cannot be interpreted as a bit-level call to an operating system, as claimed.

Examiner's response:

b) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, with respect to the Applicant's assertion that a high-level or assembly language call is not a call including a sequence of l's and 0's, the Examiner respectfully submits that one of ordinary skill in the art would readily comprehend that a high-level or assembly language call or any computer process is ultimately a "bit-level" manipulation of binary data.

Second, with respect to the Applicant's assertion that the socket function disclosed by Boucher cannot be interpreted as a bit-level call to an operating system, as previously pointed out in the Non-Final Rejection (mailed on 06/23/2008) and further clarified hereinafter, the Examiner respectfully submits that Boucher clearly discloses "wherein the socket routine reads the information from the socket through a bit-level call to an operating system" *(see Column 3: 55-67 to Column 4: 1-4, "In a first step (step 300), the Samba application program 104 initializes application-to-operating system communication by calling the "socket" function." and "The Samba application program 104 then calls the "listen" routine to wait for an incoming connection to arrive from kernel 105. When an incoming connection arrives, the Samba application program 104 calls the "accept" routine to complete the connection setup. After*

setting up the socket, the Samba application program 104 uses the "select" routine to tell the kernel 105 to alert application 104 when data for that particular connection has arrived."). Note that once the socket connection has been established, the socket function (the COBOL routine) maintains connection with the socket by interfacing with the kernel of the operating system. In other words, the socket function has to make "bit-level" calls to the operating system in order to interface with the operating system according to its kernel system call requirements. Thus, one of ordinary skill in the art would readily comprehend that in order for the socket function to interface with the operating system, it is inherent that the socket function must enable "bit-level" operating system calls, at least on the machine code level, to establish a socket connection.

Therefore, for at least the reasons set forth above, the rejections made under 35 U.S.C. § 103(a) with respect to Claims 1, 6, and 21 are proper and therefore, maintained.

Conclusion

15. The prior art made of record and not relied upon is considered pertinent to Applicant's disclosure.

16. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after

the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

17. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Q. C./

Examiner, Art Unit 2191

/Wei Y Zhen/

Supervisory Patent Examiner, Art Unit 2191